Mutua esclusione per sezioni critiche brevi

Mutua esclusione: Riepilogo dei requisiti

1. sezioni critiche eseguite con interruzioni abilitate
2. mutua esclusione dei thread nelle sezioni critiche
3. indipendenza dei thread rispetto alle sezioni critiche
4. assenza di condizioni di deadlock
5. assenza di attese attive
6. assenza di starvation

Mutua esclusione: Riepilogo dei requisiti

- I requisiti 2, 3, 4 sono di *ordine logico* e sono relativi alla *correttezza* dell'implementazione del meccanismo di mutua esclusione
- Il requisito 6 è di *ordine realizzativo* ed è ancora legato alla legato alla *correttezza* dell'implementazione
- I requisiti 1 e 5 sono di *ordine realizzativo* e non riguardano la correttezza del meccanismo ma l'*efficienza* della sua implementazione

- Il problema della mutua esclusione può essere risolto rispondendo ai soli requisiti 2, 3, 4, 6 e trascurando i requisiti 1 e 5 *solo nell'ipotesi di sezioni critiche brevi* (una soluzione parziale!)

Programmazione delle sezioni critiche

- Essere all'interno di una sezione critica costituisce per un thread uno *status privilegiato*
- Il thread ha accesso esclusivo a *dati condivisi modificabili*, e *tutti* i thread che stanno richiedendo accesso a quei dati vengono mantenuti in attesa
- Le sezioni critiche devono essere eseguite *nella maniera più rapida* possibile:
  - il thread non si deve bloccare mentre è nella sezione critica
  - la codifica deve essere efficiente (ad es., evitare la possibilità di cicli infiniti)
- Se per un malfunzionamento un thread *termina* mentre è in una sezione critica, il S.O. deve rilasciare il corrispondente vincolo di mutua esclusione, in modo che altri thread possano entrare nelle proprie sezioni critiche della classe

Schema generale per la mutua esclusione

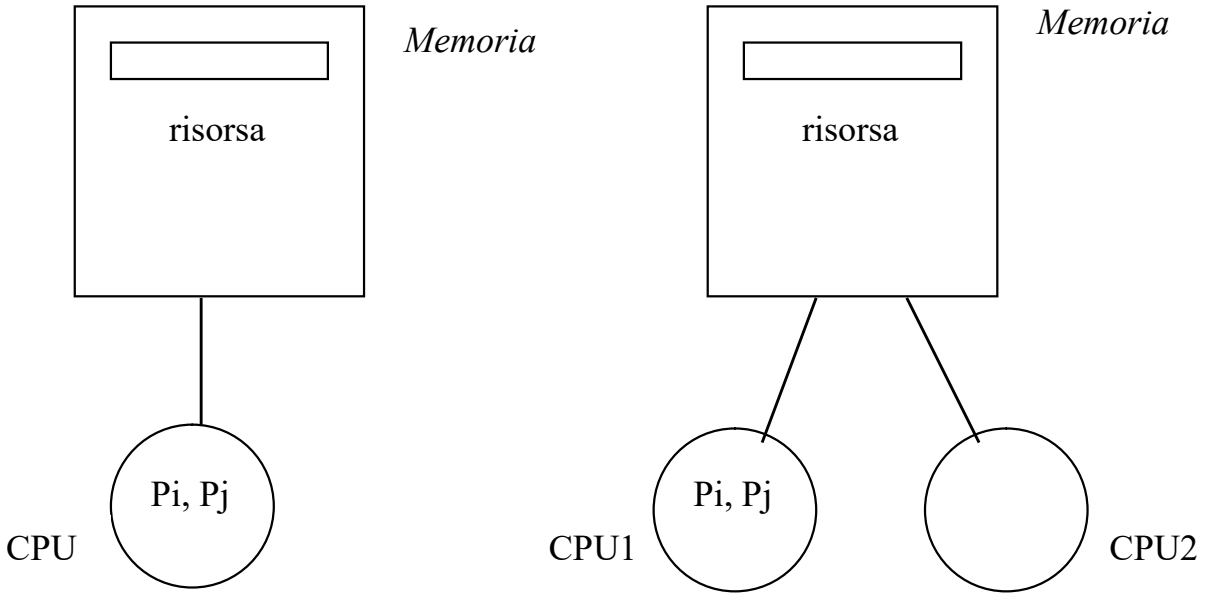
- Con ipotesi di sezioni critiche tutte brevi

Ti	Tj	Tk
...	...	...
prologo	prologo	prologo
<uso della risorsa>	<uso della risorsa>	<uso della risorsa>
epilogo	epilogo	epilogo
...	...	...

- Protocollo di accesso identico per tutti i processi/thread, idoneo per un numero arbitrario di thread, anche non noto e variabile dinamicamente

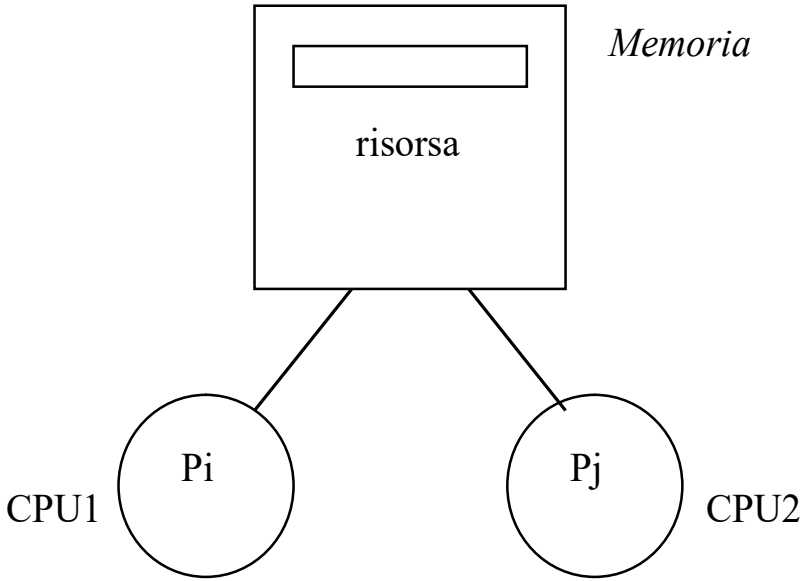
Ipotesi di sezioni critiche brevi

- {A, B, ...} classe di sezioni critiche '*sufficientemente brevi*'
- Soluzione per il caso uniprocessore:
  - Prologo: disabilitazione interruzioni
  - Epilogo: abilitazione interruzioni



Ipotesi di sezioni critiche brevi

- Soluzione per il caso multiprocessore:
- Realizzazione di prologo ed epilogo mediante le operazioni:
  - lock(x)
  - unlock(x)



lock(x), unlock(x)

```
lock(x): begin                    unlock(x): begin
           repeat until x = 1;                 x := 1;
           x := 0;                           end;
         end;
```

- x indicatore associato alla classe di sezioni critiche (inizializz. ad 1):
  - x = 1 nessuna sezione critica in esecuzione
  - x = 0 una sezione critica in esecuzione
- `lock()` e `unlock()` devono essere *indivisibili (atomiche)*
- Problema: nell'ipotesi che l'*hardware* garantisca la mutua esclusione solo a livello di singola lettura o scrittura di una cella di memoria, *solo unlock(x) è indivisibile*

Mutua esclusione mediante lock e unlock

- La mutua esclusione nell'esecuzione delle sezioni critiche appartenenti alla stessa classe si ottiene come:

Ti	Tj
...	...
lock(x)	*lock*(x)
<sezione critica>	<sezione critica>
unlock(x)	*unlock*(x)
...	...

- R2: essendo lock *primitiva* un solo thread può entrare nella sezione critica. La mutua esclusione è garantita.

Mutua esclusione mediante lock e unlock

- R3: sulla variabile x possono operare solo lock e unlock, eseguite dai thread unicamente per accedere alla sezione critica.
- R4: essendo lock e unlock primitive non possono insorgere condizioni di deadlock.
- R5: nella lock è presente una attesa attiva.
- R6: la starvation è teoricamente possibile, dipendendo dalle condizioni realizzative e non dalla soluzione logica data alla mutua esclusione.

Mutua esclusione mediante lock e unlock

- Nell'ipotesi che l'*hardware* garantisca la mutua esclusione solo a livello di singola lettura o scrittura di una cella di memoria, *solo unlock(x) è indivisibile*

- Realizzazione di lock(x):

```
LOCK:   LOAD   R1,x
        ADD    R1,R1
        JZ     LOCK
        STORE  x,R2
```

Se lock(x) non è indivisibile, si potrebbe avere:

t0:	*if* x = 1	(Pi)
t1:	*if* x = 1	(Pj)
t2:	x := 0	(Pi)
t3:	x := 0	(Pj)

→ Ti e Tj entrambi in sez. critica

- Indivisibilità mediante *inibizione delle interruzioni?* Soluzione *parziale*!

Supporto hardware

- Perchè la soluzione lock-unlock sia corretta è necessario che la lock sia *indivisibile*, cioè *primitiva*
- Normalmente l'hw garantisce la mutua esclusione *per un solo accesso alla memoria*
- Sono necessarie istruzioni e infrastrutture hardware che consentano l'*ispezione* e la *modifica* di una parola *in modo indivisibile*

- → Supporto hw: *istruzioni macchina speciali*, *arbitro*

Istruzioni speciali: test-and-set

- test_and_set(x) (IBM 360)

```
function  test_and_set (var a: boolean): boolean;
    begin
            test_and_set := a;
            a := true;              /* BUSY */
    end;
```

- copia il valore di x in un registro ed assegna ad x il valore BUSY

Istruzioni speciali: XCHG (exchange)

- XCHG (intel)

```
procedure XCHG (var a, b: boolean);
    var temp: boolean;
    begin
        temp := a;
        a := b;
        b := temp;
    end;
```

- scambia i contenuti di un registro e di una locazione di memoria in un "unico" accesso

Esempio: intel 286, multibus

THE IAPX 286 INSTRUCTION SET

XCHG—Exchange Memory/Register with Register

Opcode		Instruction	Clocks	Description
86	/r	XCHG *eb,rb*	3,mem=5	Exchange byte register with EA byte
86	/r	XCHG *rb,eb*	3,mem=5	Exchange EA byte with byte register
87	/r	XCHG *ew,rw*	3,mem=5	Exchange word register with EA word
87	/r	XCHG *rw,ew*	3,mem=5	Exchange EA word with word register
90+	*rw*	XCHG AX,*rw*	3	Exchange word register with AX
90+	*rw*	XCHG *rw*,AX	3	Exchange with word register

FLAGS MODIFIED

None

FLAGS UNDEFINED

None

OPERATION

The two operands are exchanged. The order of the operands is immaterial. BUS LOCK is asserted for the duration of the exchange, regardless of the presence or absence of the LOCK prefix or IOPL.

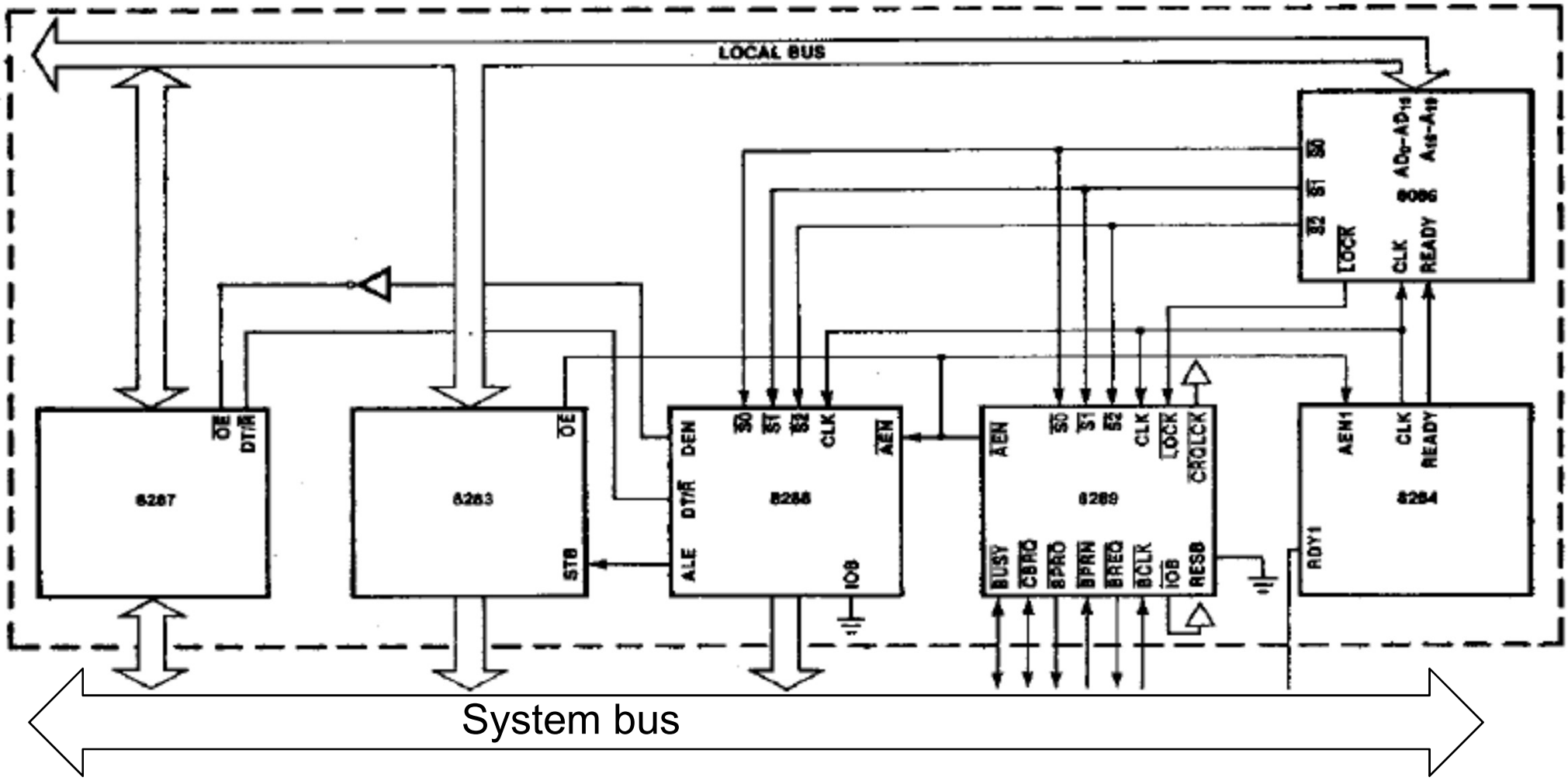
PROTECTED MODE EXCEPTIONS

#GP(0) if either operand is in a non-writable segment. #GP(0) for an illegal memory operand effective address in the CS, DS, or ES segments; #SS(0) for an illegal address in the SS segment.

REAL ADDRESS MODE EXCEPTIONS

Interrupt 13 for a word operand at offset 0FFFFH.

Esempio: intel 286, multibus



8289 = bus arbiter

Realizzazione di lock(x)

```
lock(x):    begin
                repeat until not  test_and_set(x);
            end
```

```
lock(x):    begin
                priv := true;
                repeat XCHG(x,priv);
                until  priv = false;
            end
```

Realizzazione di lock(x) con test + test and set

- lock() basata su test_and_test_and_set (TTSL), per migliore uso di cache e bus di comunicazione in sistemi multicore:

```
lock(x):    begin
                while x;
                repeat until not test_and_set(x);
            end
```

- Con TTSL, le cache locali ai core non sono invalidate inutilmente dalla operazione di scrittura dei thread che tentano di accedere quando il lock è già stato acquisito da altri thread

Mutua esclusione mediante lock e unlock

- La mutua esclusione basata su lock() e unlock() (con test_and_set sottostante) funziona senza necessità di usare le interruzioni
  - le interruzioni possono essere usate e gestite solo dal SO
  - test_and_set() – TSL è un'istruzione user-mode, disable_interrupt no!
- → lock() e unlock() possono essere usate anche in librerie o soluzioni interamente *a livello utente*
- Soluzione idonea a sistemi multiprocessore / multicore
  - se cache locali ai singoli core, meglio test_and_test_and_set() - TTSL per evitare traffico di riscrittura da cache a memoria indotto da TSL
  - TTSL prima esegue loop con solo test
- Solo per sezioni critiche brevi, prevede «spinning»!

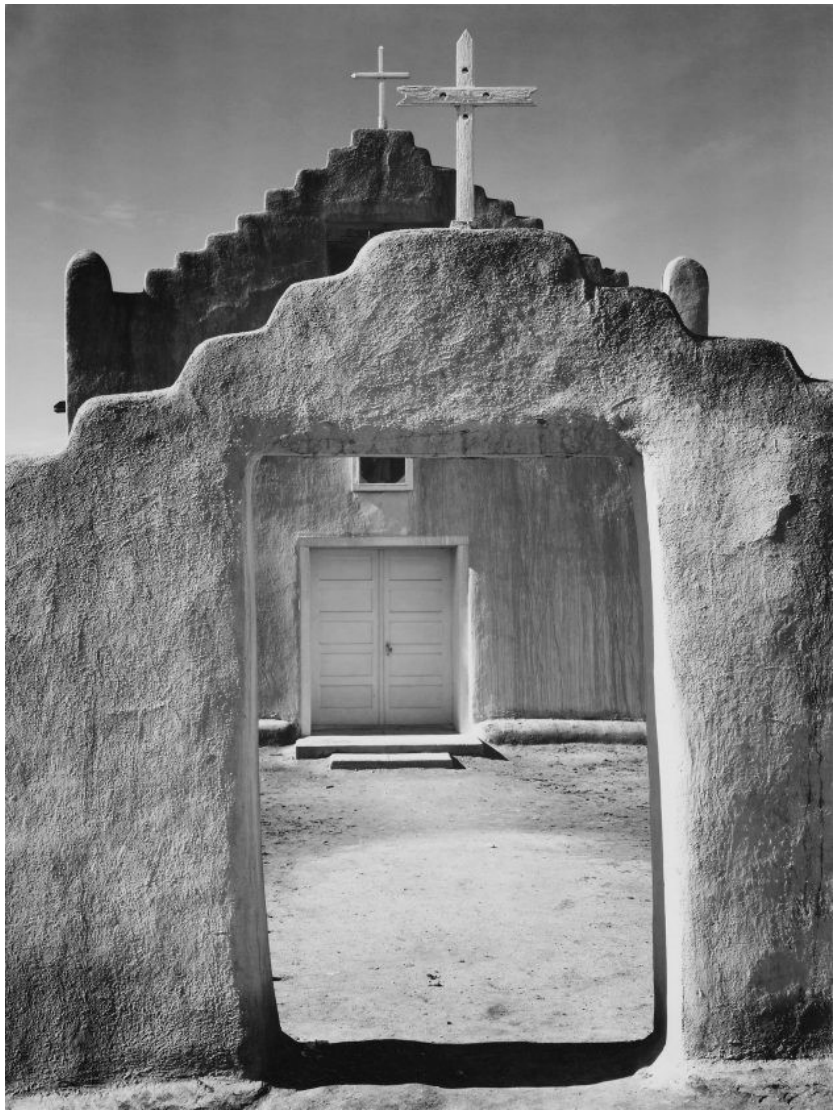
Mutua esclusione mediante lock e unlock

- Inconvenienti:
- La soluzione di per sé è valida solo per processi / thread residenti su *CPU diverse*
  - L'hw di supporto (arbitro) può garantire indivisibilità negli accessi consecutivi alla memoria centrale solo rispetto ad *altri* processori
  - In alternativa: messaggi hw di comunicazione inter-processore, lenti e complessi da realizzare e gestire
- La soluzione *è* la presenza di attese attive (spinning):
  - Il processore di Tj nell'interrogare x accede alla memoria e disturba Ti
  - Il processore di Tj è impegnato in una attività inutile: «*spinning considered harmful*» !
- La soluzione non esclude il verificarsi di fenomeni di *starvation*

Mutua esclusione mediante lock e unlock

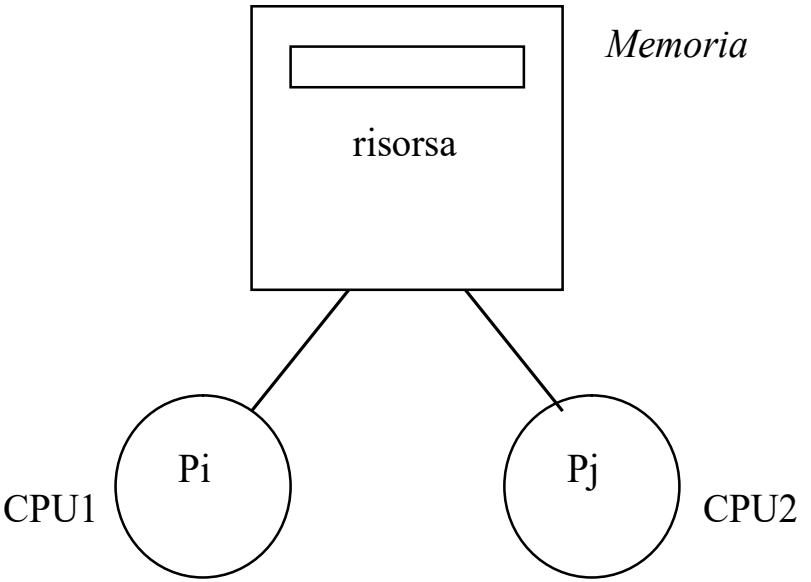
- Per rendere *più efficiente* la soluzione occorre:
  - *bloccare* un thread / processo per tutto il tempo in cui non ha accesso alla sezione critica
  - *riattivarlo* quando, per effetto del progresso di altri thread, il suo accesso alla sezione critica è consentito

Mutua esclusione con sezioni critiche brevi



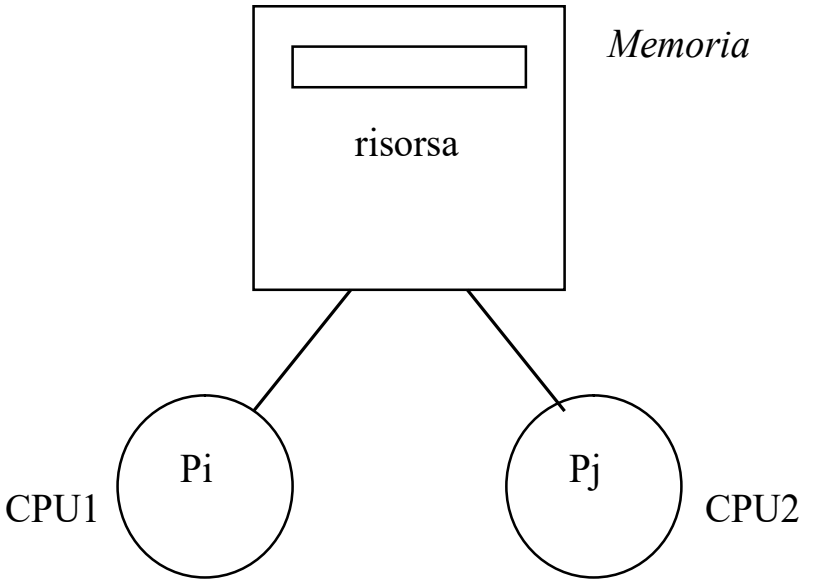
Quesito

- lock(x), unlock(x) definite per il caso multiprocessore, con 1 thread per processore
- Funzionano nel caso monoprocessore?



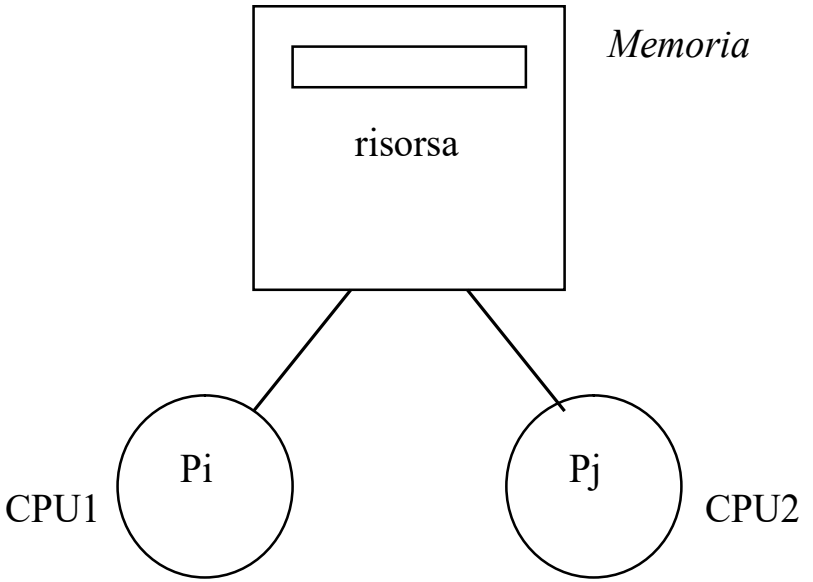
Quesito

- lock(x), unlock(x) definite per il caso multiprocessore, con 1 thread per processore
- Funzionano nel caso monoprocessore?
- Dipende!



Quesito

- lock(x), unlock(x) definite per il caso multiprocessore, con 1 thread per processore
- Funzionano nel caso monoprocessore?
- Dipende!
- Internamente c'è una test_ and_set (atomica) o c'è una sequenza assembly? Es.: -->
- Non tutti i processori supportano test_and_set!




```
OUT(Arbiter_control,LockConf)
LOAD A,x
STORE x,LockValue
OUT(Arbiter_control,UlockConf)
```

Disabilitazione interruzioni

- Se il processore non supporta test_and_set e usa sequenza di istruzioni assembly per riservare il bus, *deve disabilitare interruzioni per proteggersi da thread locali*
  - Il bus viene riservato al processore, ma sul processore ci potrebbero essere più thread in competizione!
- Altri *motivi per disabilitare interruzioni*, anche nel caso multiprocessore:
  - Esecuzione rapida di sezione critica breve, evitando disturbo da parte di thread locali che non dipendono dalla sezione critica
  - Riduce i tempi di attesa dei thread in busy waiting sugli altri processori (riduzione inversione di priorità in stile NPCS)

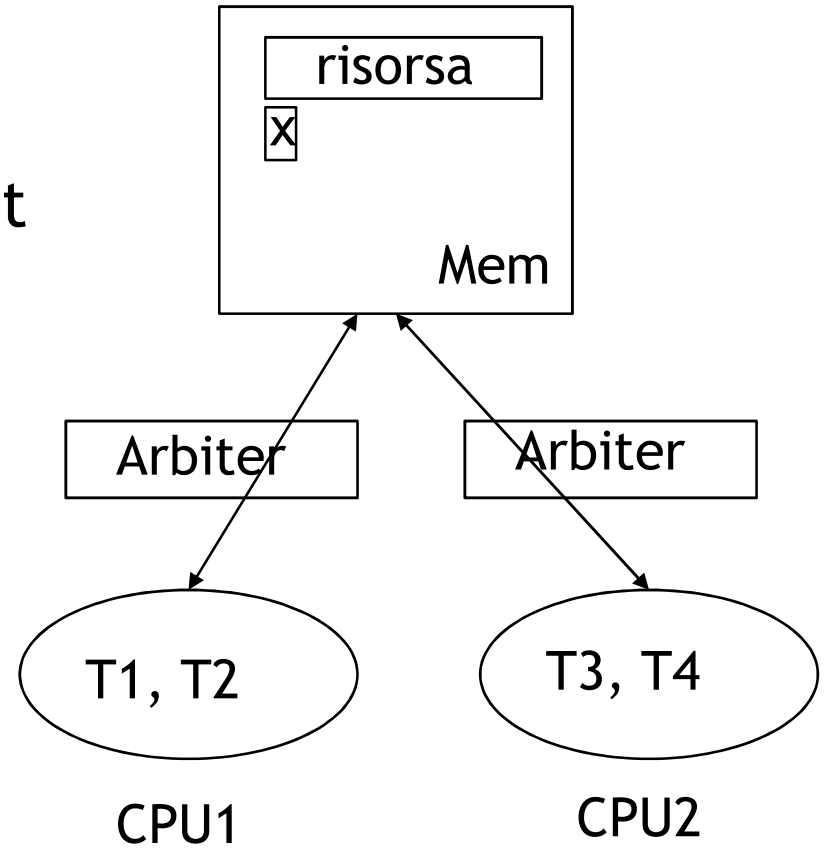
Lock / unlock e interruzioni

- Sistema multiprocessore, mutua esclusione con lock / unlock
- Thread T1, T2, T3, T4 con sezioni critiche della stessa classe, brevi
- I processori supportano test-and-set
- Ogni Ti accede con:

  lock(x)

  <sez crit>

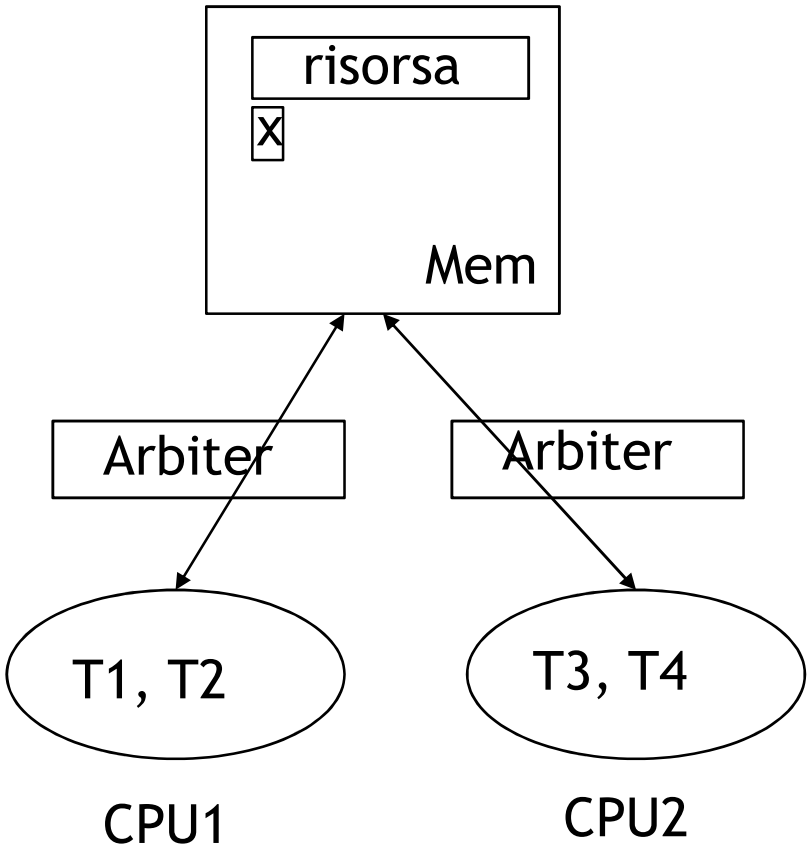
  unlock(x)



lock / unlock e interruzioni

- Possibile protocollo di accesso è:

```
Lock(x): {
        Disable;
        repeat until not test-and-set(x);
        Enable;
        }
<sez crit>
Unlock(x): {
        Disable;
        Free(x);
        Enable;
         }
```

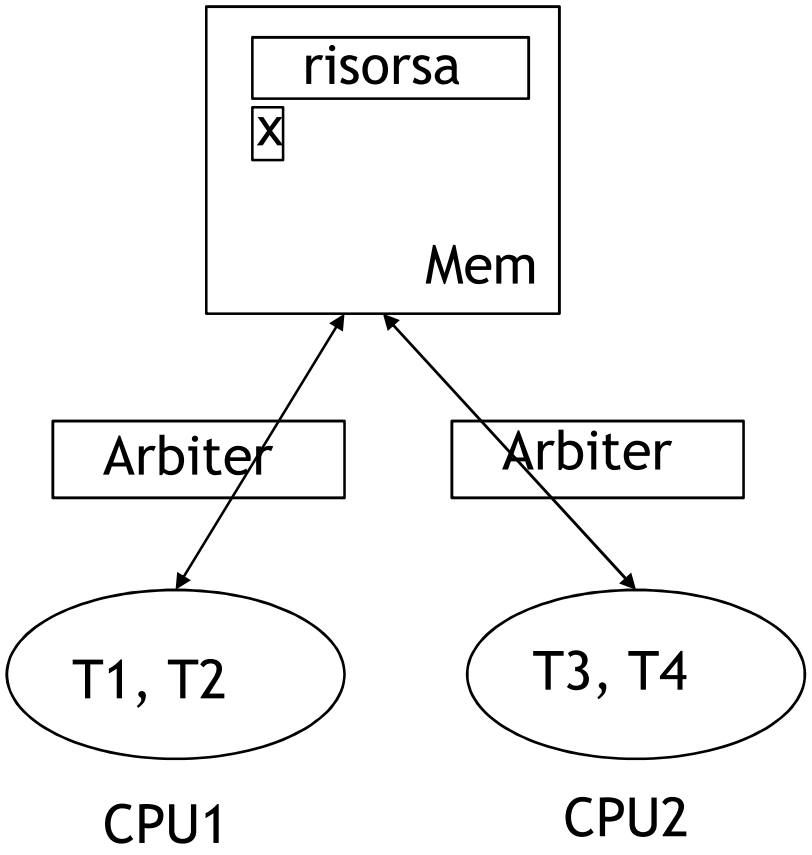


lock / unlock e interruzioni

- Possibile protocollo di accesso è:

```
Lock(x): {
        Disable;
        repeat until not test-and-set(x);
        Enable;
        }
<sez crit>
Unlock(x): {
        Disable;
        Free(x);
        Enable;
        }
```

Funziona?

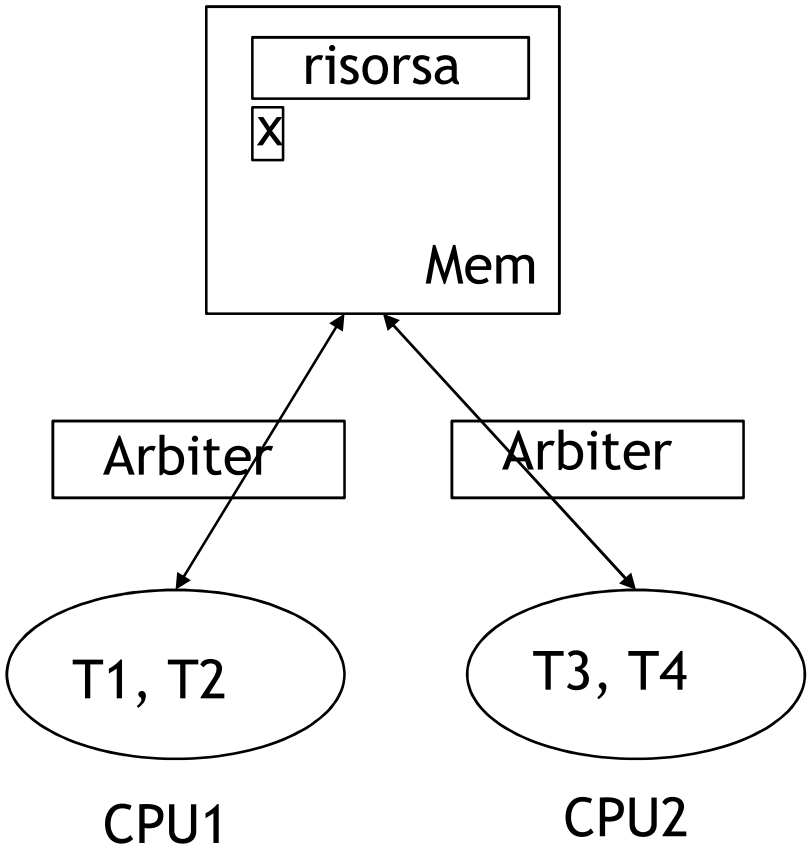


lock / unlock e interruzioni

- Possibile protocollo di accesso è:

```
Lock(x): {
        Disable;
        repeat until not test-and-set(x);
        Enable;
        }
<sez crit>
Unlock(x): {
        Disable;
        Free(x);
        Enable;
        }
```

Così funziona!



Mutua esclusione per sezioni critiche brevi: soluzione generale

- Riformulando, la soluzione generale per sezioni brevi è:

```
Disable;
Lock(x);
<sezione critica>
Unlock(x);
Enable;
```

- In sistema multiprocessore con test-and-set si può usare anche solo lock / unlock
- In sistema monoprocessore con test-and-set si può usare lock / unlock ma è preferibile disabilitare le interruzioni
- In sistema monoprocessore senza test-and-set resta la possibilità di disabilitare le interruzioni

Mutua esclusione per sezioni critiche brevi: soluzione generale

- Riformulando, la soluzione generale per sezioni brevi è:

```
Disable;
Lock(x);
<sezione critica>
Unlock(x);
Enable;
```

Spin locking vantaggioso con sezioni critiche brevi solo se la contesa è probabile derivi da thread su altri core

- In sistema multiprocessore con test-and-set si può usare anche solo lock / unlock
- In sistema monoprocessore con test-and-set si può usare lock / unlock ma è preferibile disabilitare le interruzioni
- In sistema monoprocessore senza test-and-set resta la possibilità di disabilitare le interruzioni

Mutua esclusione per sezioni critiche brevi: soluzione generale

- Proprietà:
- Nella lock(x) si può sostituire test-and-set con sequenza di istruzioni assembly, se necessario
- Adattabile a mono- e multi-processore
- Soddisfa i requisiti logici di mutua esclusione, correttezza, indipendenza sezioni critiche, assenza di deadlock
- Non determina starvation, non la previene
- Trasgredisce, in dose modica, i requisiti di efficienza: contiene busy waiting e disabilitazione delle interruzioni → solo per sezioni critiche brevi
- Non dipende dal numero di thread, non richiede conoscenza mutua dei thread, accetta thread dinamici

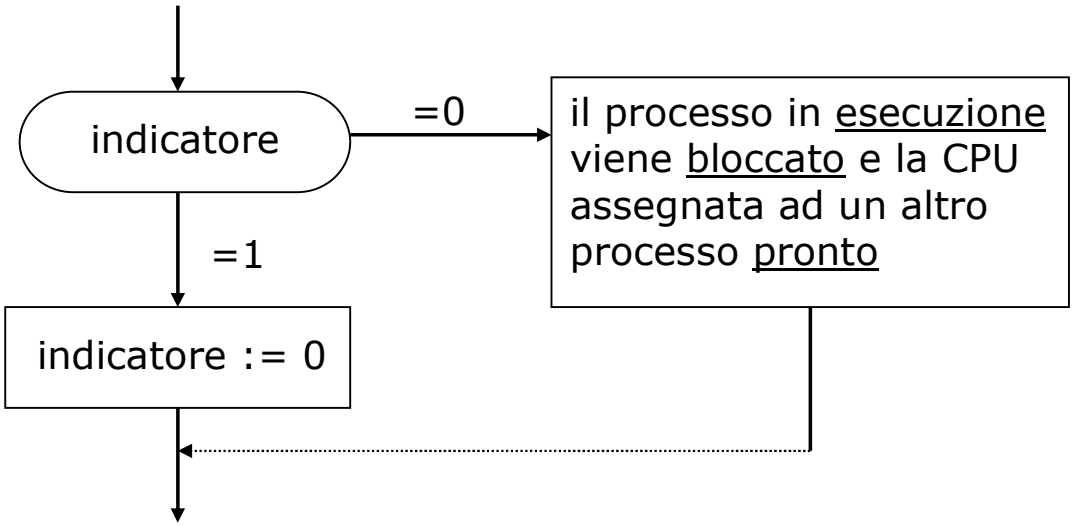
Meccanismo generale di mutua esclusione

Meccanismo di mutua esclusione generale

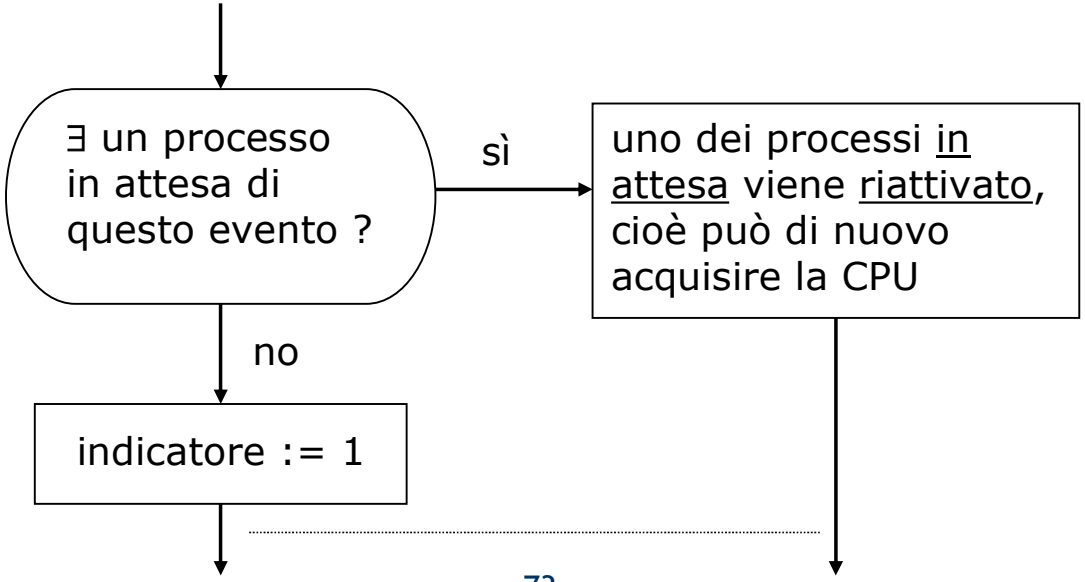
- Il meccanismo di mutua esclusione basato su lock(x) e unlock(x) e idoneo supporto hardware ha il limite di essere applicabile solo ai problemi in cui *tutte* le sezioni critiche della classe sono brevi
- Esiste un *repertorio di algoritmi per mutua esclusione* (Dekker, Peterson, Fornaio, e altri) che soffrono tuttavia dell'inconveniente della attesa attiva e fanno riferimento ad insiemi statici e mutuamente noti di processi / thread
- Quale meccanismo di mutua esclusione generale?

Meccanismo di mutua esclusione generale

- Prologo:



- Epilogo



Meccanismo di mutua esclusione generale

- *Prologo ed Epilogo devono essere operazioni 'uniche', non divisibili*
- Chiaramente, in linguaggio assemblativo prologo ed epilogo richiederanno in realtà *qualche centinaio di istruzioni*
- Come fare per renderli *atomici*?